VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Hledání cesty pro pohyb robota danými body s využitím metody regrese

Path Planning for Robot Movement Using Regression

2012 Ondřej Gronych

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Ondřej Gronych**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Hledání cesty pro pohyb robota danými body s využitím metody regrese
Path Planning for Robot Movement Using Regression

Zásady pro vypracování:

V bakalářské práci se student zaměří na hledání cesty robota pro vznikající knihovnu fotbalu robotů. Student bude v rámci práce spolupracovat s kolegy, jež se budou zabývat tvorbou taktik a analýzou obrazu s cílem vytvořit komplexní systém pro hru fotbalu robotů.

Cíle práce:
1. Nastudování metody regrese pro použití při hledání cesty danými body.
2. Vytvořit a popsat sadu experimentů pro otestování této metody.
3. Implementace této funkčnosti do části knihovny sloužící pro řízení robotů.

Seznam doporučené odborné literatury:

J. Fan and I. Gijbels: Local Polynomial Modelling and Its Applications, 1996
S. Keshmiri, S. Payandeh: Robot navigation controller: A non-parametric regression approach, Intelligent Control Systems, 2010
J. Fox: Nonparametric simple regression: smoothing scatterplots, Thousand Oaks, 2000

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Václav Svatoň**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 16. března 2012 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 16. března 2012 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Tato práce je určena ke hře fotbal robotů, kde stejně jako v klasickém fotbale, tak i v tomto by se měl umět robot vyhnout překážkám. Tato práce pomocí metody regrese naplánuje cestu robotovi tak, aby dojel úspěšně ke svému cíli a vyhnul se překážkám, které mu stojí v cestě. V práci je popsáno co je to metoda regrese a jak se počítá. Dále popisuje sadu experimentů pro ověření funkčnosti metody a její implementaci pro roboty.

Klíčová slova: regrese, polynom, robot, simulátor

Abstract

This work is intended for robot soccer football. In both classic and this the robot should be able to avoid obstacles. This work will plan robot´s path so that it succesfully reaches it´s destination and avoids obstacles using regresion method. This work describes what it is and how regression is calculated. It also describes a set of experiments to verify the functionality of the method and its implementation for robots.

Keywords: regression, polynom, robot, simulator

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

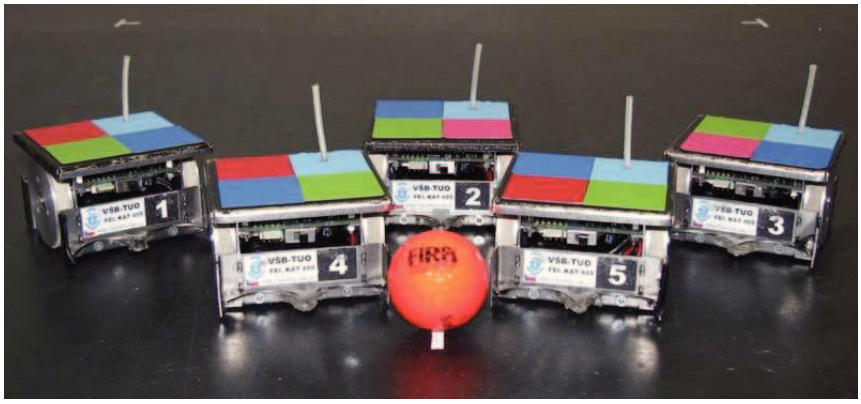
Seznam tabulek

1 Úvod

V dnešní době se robotechnika stále více rozvíjí a zdokonaluje. Mě toto téma vždy velice zajímalo a to je taky jeden z důvodů, proč jsem si vybral práci na téma hledání cesty pro pohyb robota danými body s využitím metody regrese. Jak už z názvu práce vyplývá, bude se jednat o práci s roboty a jejich pohyby, respektive jejich cestou k cíli. Tato práce je určena k vývoji hry fotbal robotů.

1.1 Fotbal robotů

Jedná se o napodobeninu klasického fotbalu, kde hlavní rozdíl je v tom, že místo hráčů ho hrají roboti. Tato hra se dělí ještě do různých kategorií, které se liší v používaných robotech.



Obrázek 1: Ukázka reálných robotů

Pro testování strategií, taktik (jsou popsány v sekci "Zaimplementování do simulátoru") a simulaci hry byl vytvořen simulátor zápasů, se kterým je spjata tato práce. Společně s kolegy se snažíme přispět svým dílem k jeho vylepšení dosavadní verze. V tomto simulátoru má hrací pole rozměry 220x180 a pohybuje se na něm pět robotů z obou týmů.

1.2 Proč plánovat cestu

Při hře dochází k mnoha situacím, kdy v cestě jednoho robota stojí například jiný robot a to vede ke srážce obou. Těchto sitací jsou až tisíce během dasítek vteřin. Představa že při reálném fotble robotů během pár desítek vteřin dojde k takovému množství kolizí je nepřípustná. Nejen, že by se roboti rychle zničili, ale v turnajích pořádaných asociací FIRA jsou dokonce tyto nehody souzeny jako faul. Dalším důvodem proč se věnovat plánování cesty je, že při cestě se robot dokáže k míči dostat rychleji a tím změnit celý průběh hry. Metod pro řešení tohoto problémů je celá řada a tato práce popisuje řešení pomocí metody regrese, neboli vytvoření cesty robota proložením polynomu libovolného stupně.

Práce je rozdělena na tři kapitoly, kde v první je vysvětlěno, co to vlastně je metoda regrese, proč je zrovna tato metoda vhodná pro řešení výše uvedeného problému a jak se s ní dá počítat a pracovat. Ve druhé části je postupná implementace této metody v samostatném projektu v jazyce C#, kde je vysvětlen a následně hned otestován každý důležitý krok implementace. V poslední třetí kapitole je přenesení kódů do samotného simulátoru zápasů, kde se navíc k tomu řeší ještě detekce kolizí.

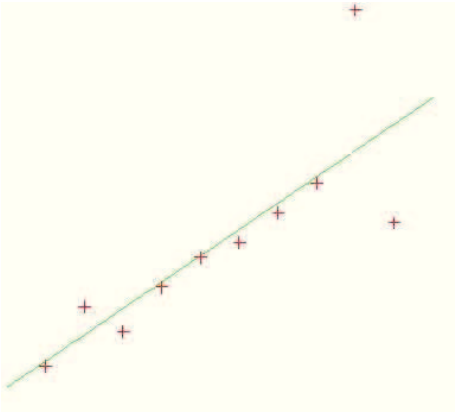
2 Metoda regrese

2.1 Úvod do regrese

V simulátoru hry trvá jeden krok několik milisekund. V každém takovém kroku jsou k dispozici informace o pozicích všech robotů a míčku. Pro zjednodušení je hrací pole rozděleno na gridové souřadnice 6x4, ale reálné souřadnice jsou 220x180. Nejdříve se provedou strategie, které pracují s gridovými souřadnicemi. Ty zjistí kde se roboti nachází a podle toho vyberou nejlepší pravidlo podle kterého řeknou robotům kam se mají pohnout (pouze do kterého gridu). Dále následuje provedení taktik, kde se analyzuje každý robot zvlášť. V úvahu se bere, zda je robot v kolizi, jestli je u míčku ap. Po provedení taktik přichází na řadu tato práce, kde jsou známy aktuální pozice všechn robotů a jejich cíle, kam se mají pohnout. Pokud robot zjistí, že mu stojí v cestě ke svému cíli jiný robot (jak už ze svého týmu, nebo od protivníka), tak ho objede. K tomuto účelu byla použita metoda regrese, která říká, že libovolnou funkci lze v omezeném rozsahu hodnot nahradit polynomem. V tomto případě, když je k dispozici pozice robota který má někam jet, je známa pozice jeho překážky, která mu stojí v cestě i jeho cíl, tak se vytvoří takový polynom a naplánuje se cesta pro robota tak, aby se vyhnul kolizi.

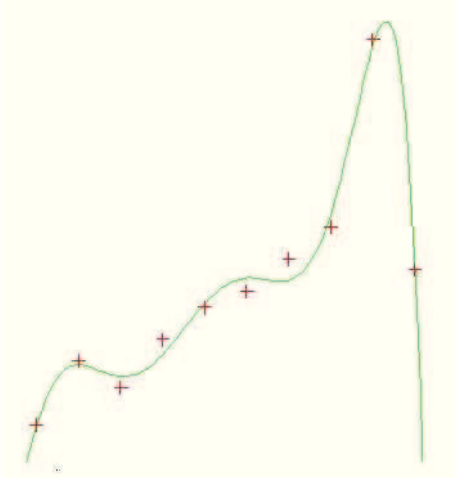
2.2 Popis regrese

Aby byl vytvořen požadovaný polynom, nemusí se vytvářet nejdříve funkce. Pomocí metody nejmenších čtverců, jejíž výpočet je vysvětlen níže, lze vypočítat polynom přímo. Dle definice metoda regrese představuje proložení (aproximaci) zadaných hodnot polynomem. Koeficienty hledaného polynomu jsou metodou nejmenších čtverců vypočteny tak, aby součet druhých mocnin odchylek původních hodnot od získaného polynomu byl minimální. Pokud polynom má stupeň roven jedné, jedná se o nejjednodušší polynom a tedy přímku. Pro ukázku, jak vypadá tedy metoda regrese v grafu byl použit polynom prvního stupně.



Obrázek 2: Ukázka lineární regrese

Na ukázce lze vidět, že prokládáním polynomem prvního stupně není přesné a tak je pro tyto potřeby nepoužitelný. S rostoucím stupněm polynomu stoupá flexibilita a zároveň prokládání bodů je mnohem přesnější. Pokud bychom ale chtěli použít tuto metodu pro predikční schopnost, tak bychom moc neuspěli. Polynom vysokého stupně je sice velice přesný, ale predikovat podle něj nelze. Na následujícím obrázku č. 3, lze vidět přesnost polynomu sedmého stupně



Obrázek 3: Ukázka polynomiálni regrese

Podle ukázky je vidět, že polynom je velice přesný, avšak predikovat podle něj nelze. To ovšem v tomto případě ani není cílem a tak se toto nemusí řešit. Pro potřeby, kde jsou známy tři body, je cílem vytvořit takový polynom, který by byl vhodný a mohli bychom poslat robota po něm. K tomuto účelu je vysoký stupeň polynomu zbytečný. Jeho výpočet by byl příliš složitý, celý běh hry by nadměrně zpomaloval. Pro vytvoření polynomu ze tří bodů stačí polynom druhého stupně. To znamená že dané body proložíme parabolou, někdy se tomuto postupu taky říká kvadratická regrese. Takovýto polynom je jednoduchý na výpočet, to znamená, že nebude nadměrně zpomalovat běh hry a je dostatečně přesný. Na obrázku č. 9 lze vidět, jak je polynom druhého stupně vytvořený třemi body přesný.

2.3 Výpočet polynomiální regrese

Výpočet polynomiální (někdy taky polynomické) regrese je pro někoho složitý, pro někoho méně. K výpočtu polynomu libovolného stupně je hlavní znát Gauss - Jordánovu eliminační mětodu, nebo jinou metodu, pro výpočet matic. To proto, že polynom libovolného stupně se dá přepsat do maticového tvaru a pomocí právě Gauss - Jordánovy eliminační metody lze vypočítat koeficienty požadovaného polynomu.

2.3.1 Výpočet obecně

Předpoklad polynomiální regrese je, že předem dané body

$$(x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_n, y_n) \tag{1}$$

lze proložit polynomem k-tého stupně

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... + a_k x^k, k < n \tag{2}$$

a takovýto polynom lze vyjádřit v následující rovnici.

$$Ax \approx b \tag{3}$$

Kde A je matice tvořená ze sum x-ových souřadnic bodů, x je náš neznámý vektor koeficientů polynomu a b je vektor tvořený ze sum x-ových a y-ových souřadnic bodů. Tuto rovnici lze přepsat následovně:

$$\begin{bmatrix} n & \sum x_i & \cdots & \sum x_i^k \\ \sum x_i & \sum x_i^2 & \cdots & \sum x_i^{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum x_i^k & \sum x_i^{k+1} & \cdots & \sum x_i^{2k} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum y_i x_i \\ \vdots \\ \sum y_i x_i^k \end{bmatrix} \tag{4}$$

Prostřední vektor tvořený členy $a_0, a_1 \cdots a_k$ jsou právě potřebné koeficienty rovnice č. 2. Tato rovnice se poté řeší pomocí Gauss - Jordánovy eliminační metody, kde je cílem získat jednotkovou matici a výsledky, které vyjdou v posledním pravém přidaném sloupci budou právě ony koeficienty.

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & a_0 \\ 0 & 1 & \cdots & 0 & a_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & a_k \end{bmatrix} \tag{5}$$

Pro další postup práce je potřeba samotnému výpočtu dobře porozumět, proto jestě samotný výpočet bude ukázán na konkrétním číselném příkladu. Ještě před samotným výpočtem je nutno upozornit, že tvar všech sum v matici v rovnici č. 4 je pouze zkratka pro

$$\sum x_i = \sum_{i=0}^{n} x_i \tag{6}$$

2.3.2 Číselný příklad výpočtu

Při počítání konkrétního příkladu je použit čtvrtý stupeň polynomu a osm bodů, ze kterých bude polynom tvořen. Čtvrtý stupeň polynomu a větší množství bodů byly zvoleny zcela záměrně a to proto, že na takovém příkladu bude dobře vidět celý výpočet a ve výsledném grafu polynomu půjde dobře vidět jeho flexibilita.

Jako vstupní data bude tato množinu bodů:

$$\{(-1,-1),(0,3),(1,2.5),(2,5),(3,4),(5,2),(7,5),(9,4)\} \tag{7}$$

Podle rovnice č. 4 lze vidět, že celá matice je tvořená ze sum x-ových souřadnic. Proto tuto množinu pro přehlednost a další výpočty je výhodné přepsat do tabulky. Později i při samotné implementaci budou body rozdělené do dvou polí, kde budou zvlášť x-ové souřadnice a zvlášť y-ové souřadnice.

X	-1	0	1	2	3	5	7	9
Y	-1	3	2.5	5	4	2	5	4

Tabulka 1: Body tvořící polynom

Jelikož sumy v matici se opakují, je vhodné si je vypsat zvlášť a nepočítat je pokaždé znovu. Proto před samotným naplněním matice je vytvořena ještě následující tabulka.

$\sum x_i^0$	$\sum x_i^1$	$\sum x_i^2$	$\sum x_i^3$	$\sum x_i^4$	$\sum x_i^5$	$\sum x_i^6$	$\sum x_i^7$	$\sum x_i^8$
8	26	170	1232	9686	79256	665510	5686952	49208966

Tabulka 2: Sumy x-ových souřadnic bodů

Nyní už není obtížné si matici č. 4 přepsat do následující podoby.

$$\begin{bmatrix} 8 & 26 & 170 & 1232 & 9686 & 24.5 \\ 26 & 170 & 1232 & 9686 & 79256 & 106.5 \\ 170 & 1232 & 9686 & 79256 & 665510 & 676.5 \\ 1232 & 9686 & 79256 & 665510 & 5686952 & 5032.5 \\ 9686 & 79256 & 665510 & 5686952 & 49208966 & 39904.5 \end{bmatrix} \tag{8}$$

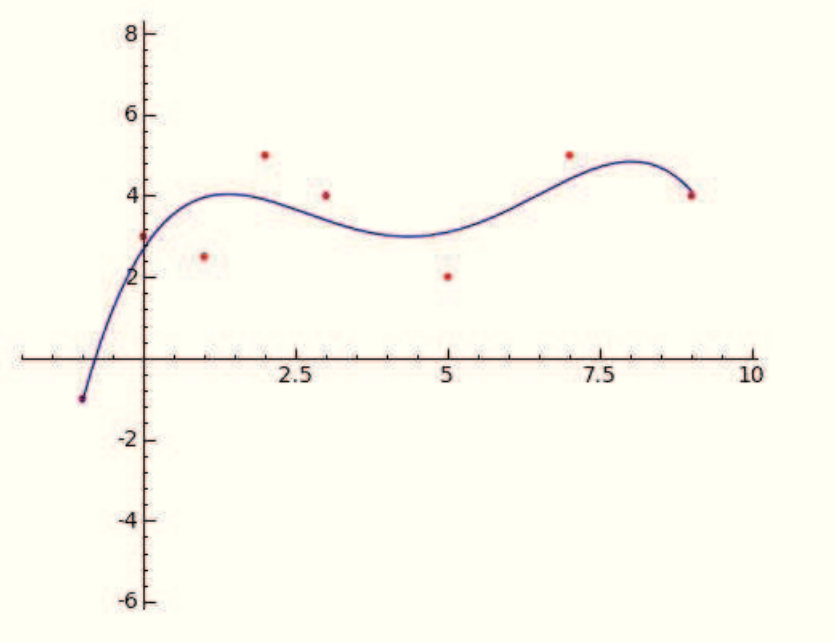
Poslední šestý sloupec je právě vektor b z rovnice č. 3. Nyní se na tuto matici provede Gauss - Jordánova eliminační metoda, kde cílem je získat na levo od přidaného sloupce matici jednotkovou.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 2.68559 \\ 0 & 1 & 0 & 0 & 0 & 2.30146 \\ 0 & 0 & 1 & 0 & 0 & -1.23263 \\ 0 & 0 & 0 & 1 & 0 & 0.216892 \\ 0 & 0 & 0 & 0 & 1 & -0.0118197 \end{bmatrix} \tag{9}$$

Výsledky, které vyšly v posledním, přidaném sloupci, jsou právě počítané koeficienty požadovaného polynomu k-tého stupně. Pro tento číselný příklad tedy bude mít polynom čtvrtého stupně následující tvar

$$y = -0.012x^4 + 0.217x^3 - 1.233x^2 + 2.301x + 2.686 \tag{10}$$

K dokončení tohoto příkladu už chybí pouze graf, na kterém lze vidět, jak je polynom flexibilní.



Obrázek 4: Graf číselného výpočtu

Je vidět, že výpočet polynomu libovolného stupně není složitý, ale pokud se jedná o polynom nějakého vyššího stupně, jeho výpočet by už zabral hodně času a pro hru fotbal robotů by se nedal použít. Představa, že v každém kroku, který trvá pár milisekund by se měl počítat pro každého robota na hřišti polynom nějakého vyššího stupně než druhého je zbytečná. Proto dále v této práci budou již všechny ukázky a samotné implementace prováděny pouze na polynomu druhého stupně tvořeného třemi body.

3 Implementace v odděleném projektu

Poté co je vysvětleno vše potřebné o vytváření polynomu z daných bodů, přichází na řadu tuto metodu naimplementovat a řádně otestovat. V této části kapitoly je vysvětlen postupný vývoj programu po částech a testováním správnosti každé této části. Jako první bude vysvětlena implementace Gauss - Jordnovy eliminace, bez které výpočet nelze provést. Testování tohoto výpočtu bude provedeno na vytvořené konkrétní matici. Poté se bude pozornost věnovat části programu, která má za úkol dle vzorce č. 4 matici naplnit. Tedy už nebude matice vytvořená staticky, ale vytvoří se sama na základě souřadnic známých bodů. Po spojení těchto dvou částí bude výsledkem rovnice požadovaného polynomu. Tato rovnice dále bude použita pro implementaci metody, která bude sloužit k získání seznamu bodů tvořících tento polynom pro celé hrací pole. V této části kapitoly bude uvedeno několik testů, kde na obrázcích půjde vidět, jak se vytvořil polynom z předem daných bodů. Celá tato část je implementována jako konzolová aplikace a tedy vykreslování polynomu bude do konzole. V závěru této kapitoly bude ještě vysvětleno, jakým způsobem se zjistí bod, který se dosadí do výpočtu polynomu, aby se cesta robota naplánovala správně a vyhnul se tak kolizi s jiným robotem.

3.1 Gauss - Jordánova eliminace

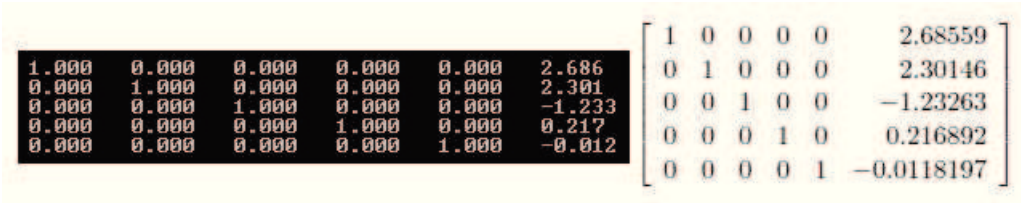
V teoretickém i číselném příkladu bylo vidět, že výpočet se provádí pomocí Gauss - Jordánovy eliminace, proto je to první věc, která by se měla naimplementovat. Ve výpisu č. 1 uvedeném níže je ukázka, jak vypadá kód výpočtu Gauss - Jordánovy eliminační metody.

```
for (i = 0; i < n; ++i){
    max = i;
  for (j = i + 1; j < n; ++j)
      if (a[j, i] > a[max, i])
        max = j;
  //prehozeni radku
  for (j = 0; j < n + 1; ++j){
      t = a[max, j];
    a[max, j] = a[i, j];
    a[i, j] = t;}
```

```
        //odecteni radku
        for (j = n; j >= i; --j)
            for (k = i + 1; k < n; ++k)
                a[k, j] -= a[k, i] / a[i, i] * a[i, j];
    }
    for (i = n - 1; i >= 0; --i){
        //vydeleni radku tak,aby na diagonale bylo cislo 1
        a[i][n] = a[i][n] / a[i][i];
        a[i][i] = 1;
        //odecteni radku
        for (j = i - 1; j >= 0; --j){
            a[j][n] -= a[j][i] * a[i][n];
            a[j][i] = 0;
        }
    }
```

Výpis 1: Implementace Gauss - Jordánovy eliminace

Kde ”n”je proměnná typu int a představuje počet řádků a sloupců a ”a”je dvourozměrné pole typu double a představuje vytvořenou matici. Výše uvedená implementace z předem dané matice vytvoří matici jednotkovou a v posledním pravém sloupci jsou potřebné výsledky. Pro otestování správnosti byla vytvořena stejná matice jako v rovnici č. 9. Na následujícím obrázku lze srovnat výsledky číselného výpočtu a implementace.



Obrázek 5: Srovnání implementace Gaussovy eliminace a číselného výpočtu

Na obrázku č. 5 je na levé straně výpis z konzole zaokrouhlený na tři desetiná místa a na pravé číselný příklad z předešlé kapitoly. Je vidět, že výsledky jsou stejné a tedy implementace pro další využití je správná.

3.2 Naplnění matice

Nyní když je správně naimplemetován výpočet matice je nutno tuto matici naplnit správnými hodnotami. Pro tuto potřebu byla naimplementována metoda createMatrix(), která

se zavolá před výpočtem gaussovy eliminace. K této metodě ještě navíc byla napsána pomocná metoda mocninaAnaN(int,int) která vrátí libovolnou mocninu libovolného čísla. Samotná implementace poté vypadá následovně.

```
double sum = 0, sumy = 0;
int moc1 = 0, moc2 = -1, pocetBodu = bodX.Count();
//vypocet sum a naplneni do matice
for (int i = 0; i < n; i++){
    moc2++;
    moc1 = moc2;
    for (int j = 0; j < n; j++){
        for (int k = 0; k < pocetBodu; k++){
            sum = sum + mocninaAnaN(bodX[k], moc1);
        }
    a[i, j] = sum;
    sum = 0;
    moc1++;
    }
}
//vypocet posledniho sloupce a jeho pridani do matice
for (int j = 0; j < n; j++){
    for (int k = 0; k < pocetBodu; k++){
    sum = sum + (mocninaAnaN(bodX[k], j) * bodY[k]);
    }
a[j, n] = sum;
sum = 0;
}
```

Výpis 2: Implementace metody pro naplnění matice

Při implementaci a tvoření matice nezáleží na pořadí bodů a proto všechny x-ové souřadnice všech bodů jsou uloženy do pole ”BodX”a jejich y-ové souřadnice do pole ”BodY”. Pro ověření správnosti naplnění matice byl opět použit číselný výpočet z výše uvedené rovnice č. 8.

8.0	26.0	170.0	1232.0	9686.0	24.5
26.0	170.0	1232.0	9686.0	79256.0	106.5
170.0	1232.0	9686.0	79256.0	665510.0	676.5
1232.0	9686.0	79256.0	665510.0	5686952.0	5032.5
9686.0	79256.0	665510.0	5686952.0	49208966.0	39904.5

$$\begin{bmatrix} 8 & 26 & 170 & 1232 & 9686 & 24.5 \\ 26 & 170 & 1232 & 9686 & 79256 & 106.5 \\ 170 & 1232 & 9686 & 79256 & 665510 & 676.5 \\ 1232 & 9686 & 79256 & 665510 & 5686952 & 5032.5 \\ 9686 & 79256 & 665510 & 5686952 & 49208966 & 39904.5 \end{bmatrix}$$

Obrázek 6: Srovnání implementace naplnění matice s číselným příkladem

Na obrázku je vidět, že maticový výpis z konzole a pod ní matice číselného výpočtu jsou totožné. Nyní když se provede již naimplementována Guss - Jordánova eliminace na tuto matici, tak výsledkem budou koeficienty onoho hledného polynomu.

3.3 Spojení podčástí a vytvoření seznamu

Pro testovací účely v samostatném projektu byly zvoleny rozměry plochy pro vykreslení polynomu na ose x od 0 do 30 a na ose y od 0 do 20. Později v simulátoru fotbalu budou rozměry jiné. Další částí je metoda, která bude vracet seznam bodů tvořících polynom. K tomu byla vytvořena pomocná třída Bod, která má dva atributy a těmy jsou právě souřadnice x a y. Později tato třída bude ještě využívaná při instanciování konkrétních robotů a míčku. Implementace metody, pomocí které lze získat list bodů vypadá takto.

```
public List<Bod> polynom(){
        List<Bod> pol = new List<Bod>();
        Bod p;
        for (int i = 0; i < 30; i++){
                p = new Bod(i,vypocet(i));
                pol.Add(p);
        }
        return pol;
}
```

Výpis 3: Metoda pro získání množiny bodů náležící polynomu

Lze si povšimnout že uprostřed cyklu for se volá metoda ”vypocet”jejím parametrem je souřadnice x. Tato metoda nedělá nic jiného, než že podle již vypočítaného polynomu k dané x-ové souřadnici vypočítá příslušnou y-ovou souřadnici. K tomu opět využívá funkci ”mocninaAnaN”. Na následující ukázce lze vidět implementaci této metody.

```
public double vypocet(int x){
        double y = 0;
        for(int i = n - 1; i >= 0; i--)     {
                y += (a[i, n] * mocninaAnaN(x, i));
        }
        return y;
}
```

Výpis 4: Metoda pro výpočet souřadnice Y k dané souřadnici X

Metoda ”polynom”, jenž je v ukázce č. 3, bude velmi užitečná nejen pro samostatný projekt, kde každý bod v listu jednoduše bude vykreslen do konzole a lze si tím vizuálně

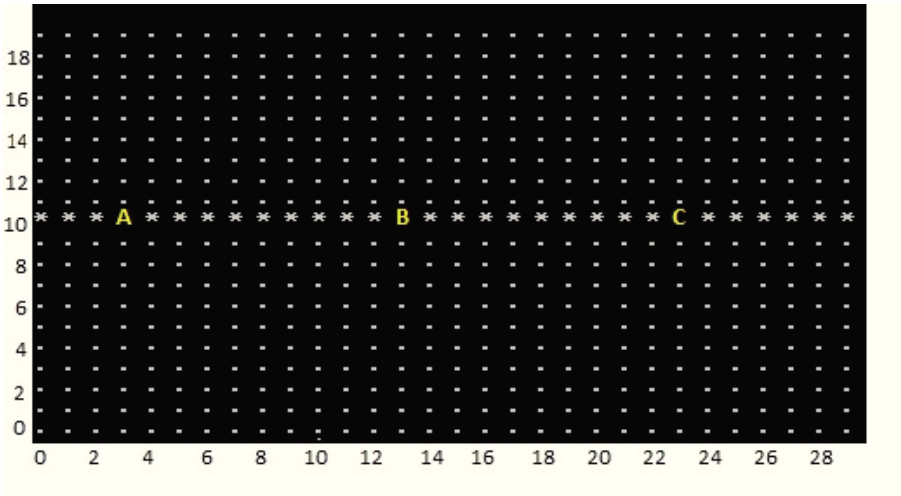
ověřit správnost, ale taky do připraveného simulátoru, kde podle x-ové souřadnice robota se zjistí jeho následující bod, kam se má pohnout.

3.4 Testování

Nyní, když už lze danými body proložit polynom libovolného stupně, přichází na řadu další fáze. Tou je sada testů, jimiž se zjistí, zda jsou výpočty opravdu správné. Otestování každé části je provedeno vždy na jejím konci, nyní tedy tyto části jsou spojeny a ověřena jejich správnost. Všechny testy nyní budou vykreslovány z konzole, kde znak ”.”označuje každý bod hracího pole a znakem ”*”jsou vykresleny body, jenž tvoří polynom. Do obrázků jsou ještě dodatečně zapsány písmem body a očíslované osy pro lepší orientaci. Prvním testem je pokus o vykreslení polynomu prvního stupně (přímky) danými body. Vstupními daty jsou následující body.

$$A[3, 10], B[13, 10], C[23, 10] \tag{11}$$

Lze vidět, že y-ové souřadnice všech tří bodů jsou stejné a tedy přímka by měla být rovnoběžná s osou x a osu y protínat v bodě 10.

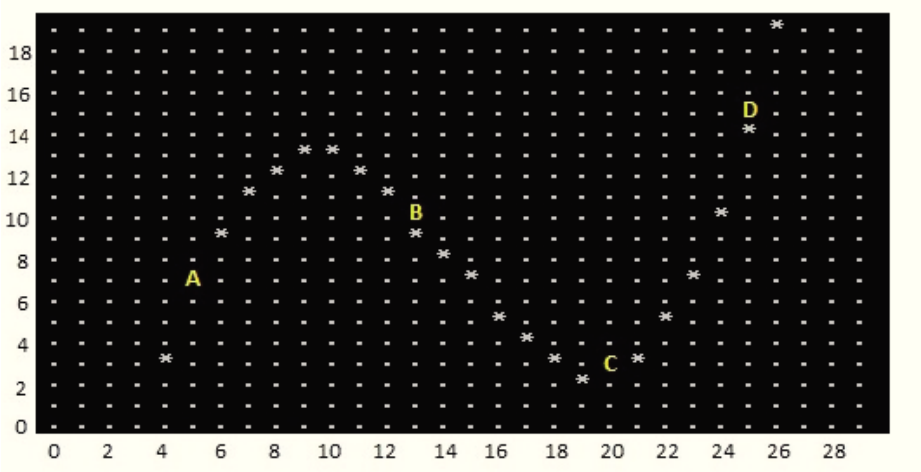


Obrázek 7: Test polynomu prvního stupně

Vytvoření polynomu prvního stupně tvořený třemi body funguje správně, ale výše uvedená implementace by měla umět vytvořit polynom libovolného stupně tvořený z libovolného množství bodů. Proto následující test bude proveden na polynomu čtvrtého stupně a bude tvořen čtveřicí následujících bodů.

$$A[5,7], B[13,10], C[20,3], D[25,15] \tag{12}$$

Podle teorie je polynom tohoto stupně pěkně "flexibilní"a měl by procházet danými body. Ve výsledku tohoto testu je možno se o tom přesvědčit.



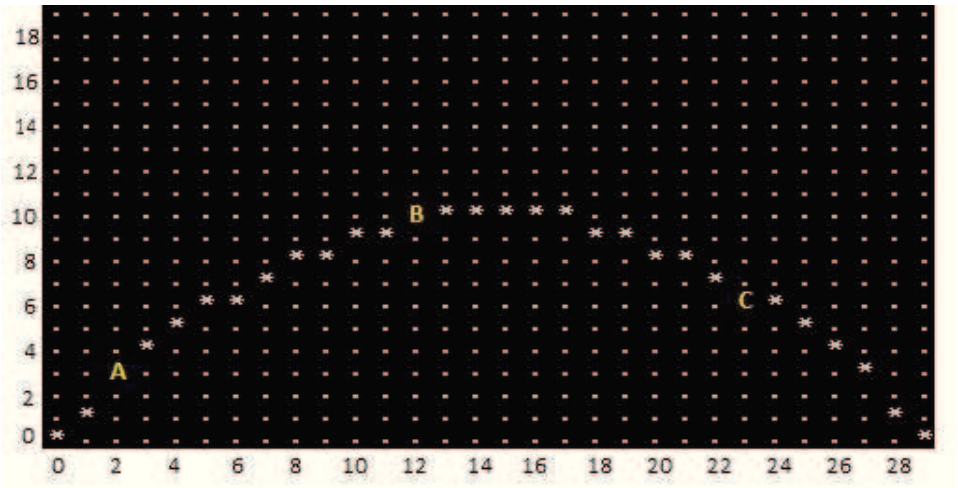
Obrázek 8: Test polynomu čtvrtého stupně

Nyní bylo ověřeno, že je implementace správná a tvoří polynom jaký byl očekáván. Polynom čtvrtého stupně opravdu prochází všemy čtyřmi zadanými body. Jenže pro fotbal robotů je toto zbytečné, protože jeden tik hry trvá pár milisekund a za tu dobu se robot pohne jen o malou část. Proto bude stačit tvořit jenom polynom druhého stupně, pomocí kterého překážku bude objíždět. Ten bude tvořen třemi body. Jeden bod bude představovat robot, který se má někam pohnout, druhý bod bude vypočítaný (výpočet bude vysvětlen v další části) podle pozice překážky a třetí bod bude místo, kam se má v plánu pohnout. Cílem dalšího testu bude zjištění, zda opravdu bude polynom druhého stupně (parabola) stačit k tomu, aby robot dokázal objet překážku a zda bude dostatečně přesný.

K tomuto pokusu byly použity následující tři body

$$A[2,3], B[12,10], C[23,6] \tag{13}$$

Výpis z konzole má poté následující podobu.

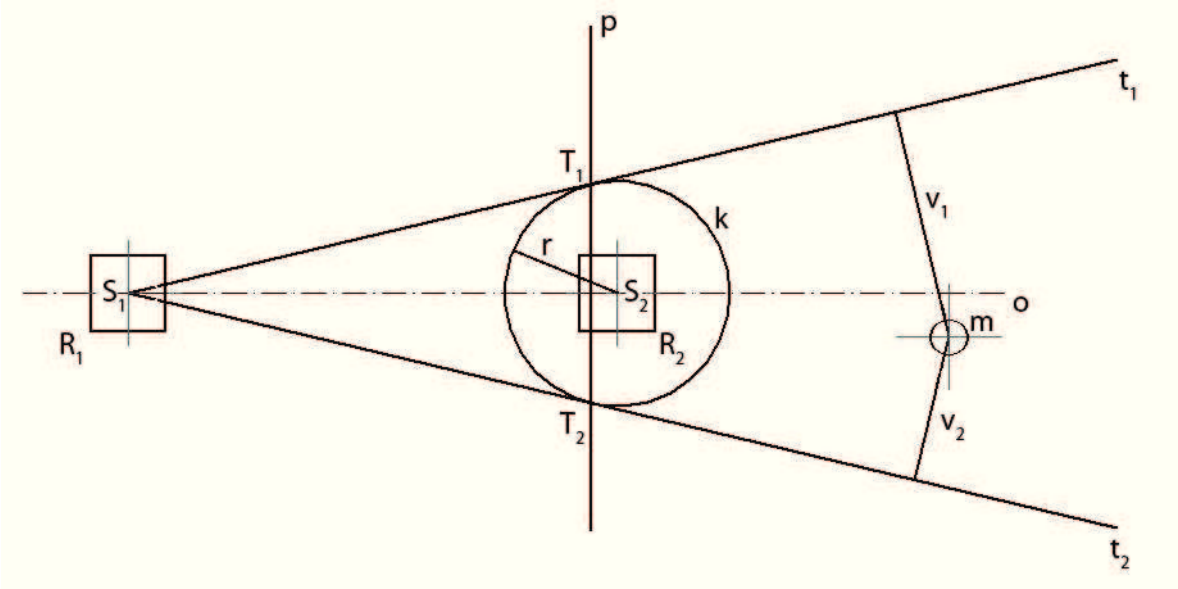


Obrázek 9: Test polynomu druhého stupně

Výsledek je uspokojivý a teoretická úvaha o tom, že by měl stačit polynom druhého stupně se potvrdila.

3.5 Zjistění bodu dotyku

V simulátoru hry jsou známy předem pozice všech robotů a míčku. Pokud ale robot zjistí, že mu v cestě stojí překážka, musí se jeho cesta naplánovat tak, aby se zabránilo kolizi. K tomu sice slouží právě regresní metoda, ale aby moha být použita je potřeba zjistit, kterým bodem bude překážku objíždět a ten do výpočtu posléze dodat. K tomuto výpočtu je použita analytická geometrie, kde se kolem překážky vypočítá kružnice o vhodném poloměru a k této kružnici se poté vedou dvě tečny středem robota. Tím vzniknou body dotyku, ze kterých se musí vybrat ten správný. Na obrázku č. 10 je ukázáno, jak se takové tečny sestrojí a jak se poté vybere vhodný bod dotyku.



Obrázek 10: Výběr bodu dotyku

Na obrázku robot R_1 se středem S_1 chce jet k míči, jenž tvoří kružnice m, ale v cestě mu stojí robot R_2 se středem S_2. Řešením, jak získat ideální bod, který by mohl být dosazen do výpočtu polynomu je následující. Kolem robota R_2 se vytvoří kružnice k o takovém poloměru r, aby se robot R_1 s robotem R_2 nesrazili. Poté z bodu S1 jsou vedeny tečky k této kružnici k. Tím vzniknou dva body dotyku T_1 a T_2. Jeden z těchto bodů bude dosazen do výpočtu polynomu. K tomu aby bylo jisté, který z těchto dvou bodů vybrat, se počítá vzdálenost bodu od přímky. Nejdříve se vypočítá vzdálenost v_1 od přímky t_1 ke středu kružnice m, poté se vypočitá vzdálenost v_2 od přímky t_2 ke středu kružnice m. Následně se tyto dvě vzdálenosti porovnaji a pokud je $v_1 < v_2$, bod který je přidán do výpočtu bude T_1. V opačném případě bude přidán bod T_2. Na obrázku č. 10 si lze ještě povšimnout, že pro zjištění bodu dotyku byla použita zvláštní přímka p, jenž se nazývá polára.

Polára p bodu $S_1[x_1, y_1]$ vzhledem ke kružnici k se středem $S_2[m, n]$ a poloměrem r obsahuje body dotyku tečen kružnice k, procházející bodem S_1. Tato přímka je dána rovnicí

$$(x - m)(x_1 - m) + (y - n)(y_1 - n) = r^2 \tag{14}$$

Z této rovnice se vyjádří x a dosadí se do rovnice kružnice. V kódu toto vyjádření a následné dosazení vypadá takto.

```
//vytvoreni rovnice polary a vyjadreni x
x1 = bodX[0] − bodX[1];
//vyjadreni polary ve tvaru x= v0y+ v1
v0 = (bodY[1] − bodY[0]) / x1;
v1 = (bodX[0] * bodX[1] − bodX[1] * bodX[1] + bodY[0] * bodY[1] − bodY[1] * bodY[1] + r * r) / x1;

//vytvoreni kvadraticke rovnice po dosazeni polary do kruznice
m = v1 − bodX[1];
a1 = v0 * v0 + 1;
b1 = (2 * v0 * m) − (2 * bodY[1]);
c1 = m * m + bodY[1] * bodY[1] − r * r;
```

Výpis 5: Vyjádření poláry a dosazení do kružnice

Proměnné a_1, b_1, a c_1 jsou koeficienty kvadratické rovnice, pro kterou je v projektu naimplementována metoda "diskriminant". Z této kvadratické rovnice jsou vypočítány y-ové souřadnice obou bodů dotyků. K nim jsou potom pomocí již vyjádřené poláry dopočítány souřadnice x-ové a tím tedy získány oba body dotyku T_1 a T_2. Z těchto bodů se poté vytvoří dvě tečny t_1 a a t_2 procházející bodem S_1. Nyní se vypočítá vzdálenost v_1 středu kružnice m od tečny t_1 a vzdálenost v_2 středu kružnice m a tečny t_2. Implementace této části v projektu poté vypadá takto.

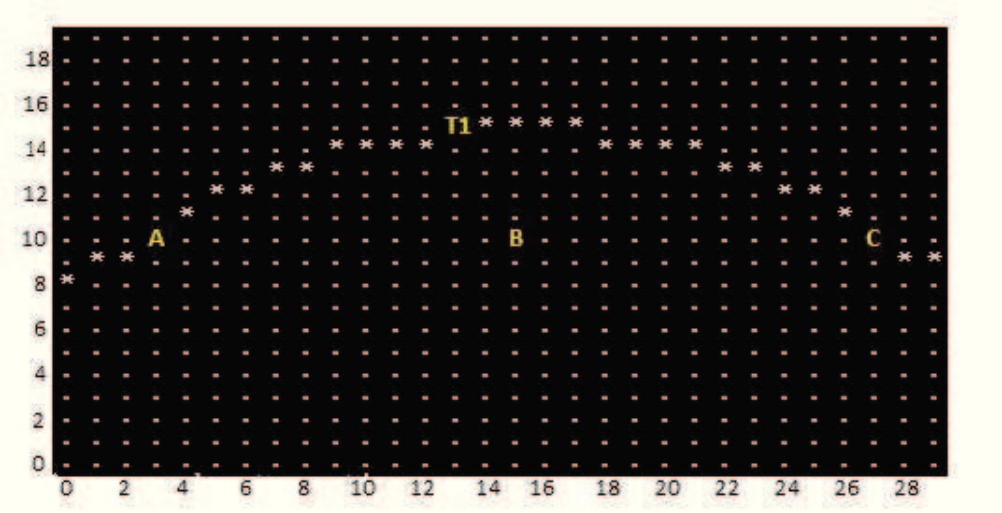
```
//vytvoreni rovnice tecen t ve tvaru a1x + b1y + c1 =
b1 = poleX[0] − bodX[0];
a1 = poleY[0] − bodY[0];
c1 = (−1 * a1 * bodX[0]) − (b1 * bodY[0]);
v0 = (a1 * bodX[2] + b1 * bodY[2] + c1) / (Math.Sqrt((a1 * a1) + (b1 * b1)));
b1 = poleX[1] − bodX[0];
a1 = poleY[1] − bodY[0];
c1 = (−1 * a1 * bodX[0]) − (b1 * bodY[0]);
v1 = (a1 * bodX[2] + b1 * bodY[2] + c1) / (Math.Sqrt((a1 * a1) + (b1 * b1)));
```

Výpis 6: Výpočet vzdáleností bodů

Tyto vzdálenosti v_1 a v_2 se porovnají a výsledný bod dotyku se přidá do výpočtu polynomu. Nutno dodat, že pokud vzdálenosti v_1 a v_2 jsou si rovny, vybere se náhodně jeden bod a při jeho vybírání se ještě musí zjistit, zda neleží mimo hrací pole. Při testování výše uvedené teorie a kódů byly zvoleny následují parametry.

$$A[3, 10], b[15, 10], c[27, 10], r = 5 \tag{15}$$

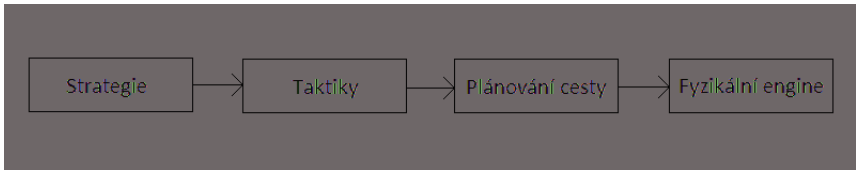
Bod A představuje robota který překážku má objet, B je robot, jenž tvoří překážku a bod C je cíl robota A. Všechny tři body mají stejné y-ové souřadnice a tedy doteď by proložení polynomem vznikla přímka. Po použití této metody se najde bod dotyku a proloží se polynom tímto bodem. Výsledek tohoto testu a také celé kapitoly je následující.



Obrázek 11: Objetí překážky

4 Zaimplementování do simulátoru

V předcházející kapitole bylo naimplementováno proložení polynomu a objetí překážky. Nyní přichází na řadu tento kód zanést do simulátoru a přizpůsobit ho robotům. V něm jsou dva typy souřadnicového systému. Pro výběr strategií a rychlou orientaci ve hře je hrací pole rozděleno do gridových souřadnic 6x4. Z takovéhoto souřadnicového systému by se kód provádět nemohl a proto bude využívat informace robotů o reálných souřadnicích. Hrací pole v tomto souřadnicovém systému má rozměry 220x180, tedy mnohem přesnější. Před zaimplementováním do simulátoru bylo potřeba se s ním seznámit a zjistit, kde výpočet regrese bude vlastně probíhat a odkud vůbec brát informace o rozložení všech robotů a míčku. Simulátor je rozdělen do několika částí, jenž se provádějí postupně za sebou. Nejdříve se provádějí strategie, které pracují v gridových souřadnicích. Ty zjistí, kde se robot nachází a "řeknou", do kterého gridu se mají v příštím kroku roboti pohnout. Poté se provádí taktiky. Ty řeší události ve hře jako jsou přihrávka, útok, obrana a podobně. V poslední řadě se provádí fyzikální engine, který pohybuje s roboty v simulátoru.

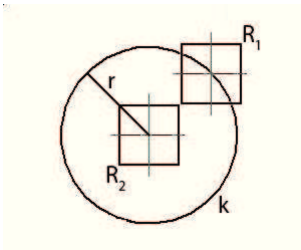


Obrázek 12: Umístění implementace v simulátoru

V této kapitole kromě zaimplementování již vytvořených tříd pro objíždění překážky bude ještě řešena detekce této překážky. Celá práce se bude provádět až před provedením fyzikálního enginu a všechny potřebné informace budou čerpány z instance třídy Storage. Ta v simulátoru představuje úložiště informací o všech objektech v hracím poli. V tomto úložišti jsou roboti rozděleni do dvou listů instancí třídy Robot. V instanci storage je pozice každého robota určena vectorem2D position, který má dva parametry X a Y. Dále v této instanci je pozice, kam se má robot pohnout, dána opět vectorem2D a nazývá se PositionMove.

4.1 Implementace pro jednoho robota

Tato část podkapitoly má za cíl pro jednoho konkrétního robota zaimplementovat metodu pro objíždění překážky. Do simulátoru byly přeneseny kódy z třetí kapitoly a bylo na nich provedeno několik malých změn. První změnou byla změna velikosti hrací plochy, která na ose x je od -110 do +110 a na ose y od -90 do +90, celkově tedy těch 220x180. Další úpravou, jež byla provedena, byl poloměr při zjišťování bodu dotyku, který se má následně dosadit do výpočtu.



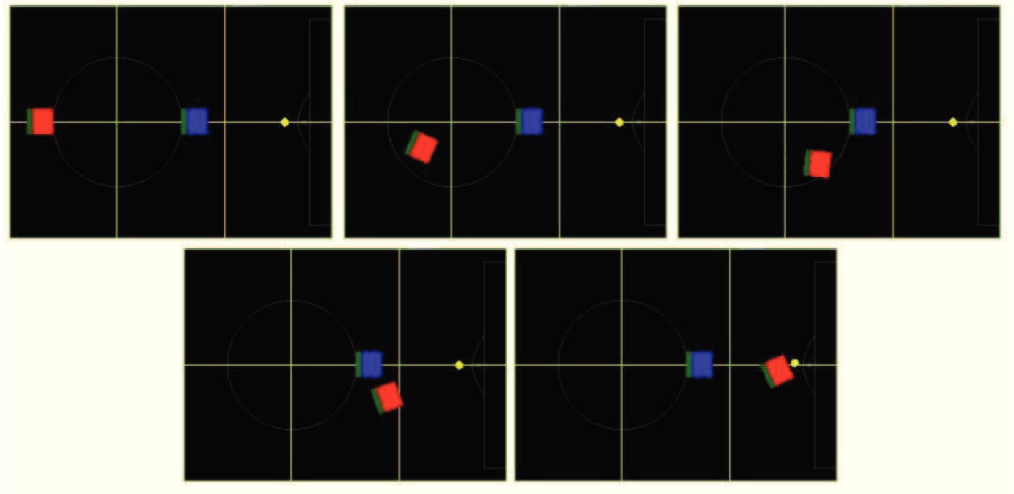
Obrázek 13: Zjištění poloměru

Dle obrázku lze vidět, že poloměr je volen tak, aby roboti do sebe nenarazili. Velikost strany jednoho robota je rovna 10. Tedy Pythagorovou větou se spočítá velikost úhlopříčky, která je v tomto případě 14,142. Aby robot měl rezervu při objíždění, tak velikost poloměru je o trochu větší. V následující ukázce je úryvek kódu, jak vypadá implementace v simulátoru pro jednoho robota.

```
//instanciování robotu
robot = new Bod(storage.MyRobots[0].Position.X, storage.MyRobots[0].Position.Y);
prekazka = new Bod(storage.OppRobots[0].Position.X, storage.OppRobots[0].Position.Y);
cil = new Bod(storage.Ball.Position.X, storage.Ball.Position.Y);
//vypocet polynomu
PolynomialRegresion regrese = new PolynomialRegresion(robo, prekazka, cil);
List<Bod> body = regrese.polynom();
//nastaveni nasledujici pozice robota
if (cil.x > robo.x)
    storage.MyRobots[0].PositionMove = new Vector2D(body[Convert.ToInt32(robo.x + 1) + 110].x,
        body[Convert.ToInt32(robo.x + 1) + 110].y);
else
    storage.MyRobots[0].PositionMove = new Vector2D(body[Convert.ToInt32(robo.x - 1) + 110].x,
        body[Convert.ToInt32(robo.x - 1) + 110].y);
```

Výpis 7: Implementace pro jednoho robota

Z počátku se vytvoří tři instance třídy Bod, které představují tři objekty v hracím poli. Poté se vytvoří instance třídy PolynomialRegresion, která obsahuje veškeré výpočty pro vytvoření polynomu ze zadaných bodů, které jsou předány v konstruktoru. Následně na to se přes metodu ”polynom”naplní seznam bodů, které polynom tvoří. Podmínka je zde proto, aby se zjistilo, zda brát z listu bod předchozí, či následující. To záleží, zda míček je na pravé straně robota či naopak. V následující ukázce je proveden test této implementace a simulátor v tomto případě je upraven tak, že výpočet strategií a taktik se vůbec neprovádí a pouze se testuje na jednom robotovi, který má přímo určenou překážku a cíl, zda jí dokáže objet. Výsledek se skládá z posloupnosti za sebou jdoucích pěti obrázků, které simulují pohyb robota při objíždění překážky.



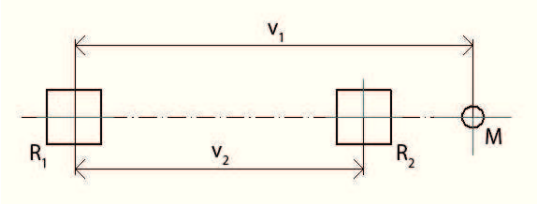
Obrázek 14: Objetí překážky

Lze vidět, že robot opravdu ve své cestě za míčem překážku dokáže objet.

4.2 Nalezení překážky

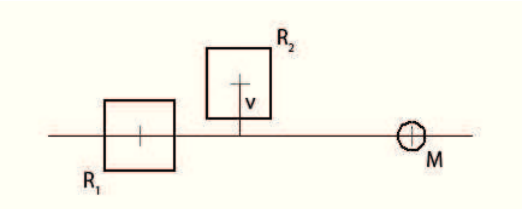
Dosud byla překážka robotu vždy přiřazena a on s ní pak pracoval. Při běhu hry ale musí detekce překážek probíhat automaticky a proto byla naimplementována pomocná metoda IsObstacle, která má za úkol zjistit, zda v cestě robota za cílem stojí překážka či nikoli. Tato metoda má tři vstupní parametry jimiž jsou instance třídy Bod a vrací pouze hodnoty

typu boolean. Na následujících obrázcích je ukázáno co všechno musí splňovat robot, aby se stal překážkou jinému robotovi v cestě. První podmínkou je vzdálenost objektů na hracím poli.



Obrázek 15: Zjištění překážky, podmínka první

Aby robot R_2 mohl být překážkou musí vzdálenost v_2 být menší než vzdálenost v_1. Tzn., že překážka musí být blíže než cíl. Ovšem to samo o sobě nestačí, protože robot R_2 by se mohl nacházet blízko za robotem R_1 a podmínka by byla vyhodnocena jako pravdivá, i když by tomu tak nebylo. Proto dodatek k této podmínce je, že obě x-ové souřadnice překážky a cíle musí být větší než souřadnice x robota nebo obě menší(záleží zda cíl je nalevo od robota nebo napravo). Pro zjištění vzdálenosti dvou bodů byla naimplementována pomocná metoda distanceTwoPoints, která podle vzorce vzdálenost vypočte a vrátí. Další podmínkou, která je nutná k tomu, aby byla nalezena překážka, je vzdálenost bodu od přímky. Protože robot není v simulátoru pouze bod, ale představuje objekt ve tvaru čtverce, musí se zajistit, aby do sebe roboti nenarazili, i když by neleželi na stejné přímce.



Obrázek 16: Zjištění překážky, podmínka druhá

Řešením této podmínky je proložit robotem R_1 a jeho cílem M přímku. Poté se zjišťuje, jestli vzdálenost v robota R_2 od přímky je menší než možná přípustná. Pokud platí první

i druhá podmínka, robot R_2 je brán za překážku. V simulátoru je tato metoda pro zjištění překážky implementována takto.

```
b = cil.x − robo.x;
a = (cil.y − robo.y) ∗ (−1);
c = −1 ∗ (a ∗ robo.x + b ∗ robo.y);
v = (a ∗ prekazka.x + b ∗ prekazka.y + c) / (Math.Sqrt((a ∗ a) + (b ∗ b)));
if (v < 0) v ∗= −1;
if (v < 15 && distanceTwoPoints(robo, prekazka) < distanceTwoPoints(robo, cil)
&& ((prekazka.x > robo.x && cil.x > robo.x) || (prekazka.x < robo.x && cil.x < robo.x))){
    return true;
    }
else return false;
```

Výpis 8: Metoda pro zjištění překážky

Otestování této metody bylo provedeno opět na jednom robotovi a ten úspěšně vyhodnotil v cestě překážku. Tu následně objel a dojel k míči, výsledek byl totožný jako na obázku č. 14.

4.3 Zaimplementování pro všechny roboty

Závěrem této práce je zaimplementování všech dosud vytvořených metod a výpočtů pro všechny roboty na hracím poli. Původně při implementaci této části vznikali problémy, kdy robot začal objíždět překážku, ale před tím narazil do jiné. Bylo to způsobené tím, že i když metoda pro zjištění překážky je už naimplementována, neřeší problém co se stane, pokud v cestě stojí překážek více. K tomuto účelu byla vytvořena pomocná třída RobotDistance, jejíž konstruktor má dva parametry. Prvním je instance třídy Bod, ta představuje samotnou překážku a druhá, typu double, je pro uchování vzdálenosti robota od této překážky. V kódu se potom pro každého robota otestují všichni ostatní roboti. At už ze stejného týmu, nebo protivníkova a pokud mu v jeho cestě za cílem tento robot překáží, přidá se do listu překážek, který je typu právě RobotDistance. Ještě před přidáním do seznamu překážek se spočítá vzdálenost robota od překážky pomocí již naimplementované metody distanceTwoPoints. Poté se tento list setřídí podle vzdáleností. Tím se zajistí, že nezačne objíždět třeba až třetí překážku v cestě, ale hned první, která je nejblíže. Jelikož jeden tik herního času trvá pouhých 80ms a v každém se provádí

výpočet polynomu (pokud dojde k nalezení překážky), je zbytečné uvažovat překážek více najednou a hledat nejlepší cestu k cíli mezi nima. Musel by k tomu být použit vyšší stupeň polynomu, tím by došlo ke značnému zpomalení hry, protože s každým zvýšeným stupněm, dochází ke značně složitějším výpočtům. Robot se na hřišti za 80ms pohne pouze o malý kousek a stačí tedy, když pokaždé najde nejbližší překážku a té se následně vyhne. K tomuto stačí druhý stupeň polynomu, jehož výpočet je rychlý a hru zbytečně nezpomaluje. Níže uvedený výtažek z kódu ukazuje nalezení nejbližší překážky.

```
List<RobotDistance> lis = new List<RobotDistance>();
for (int j = 0; j < 5; j++){
        robot = new Bod(MyRobots[j].Position.X, MyRobots[j].Position.Y);
        cil = new Bod(MyRobots[j].PositionMove.X, MyRobots[j].PositionMove.Y);
        for (int i = 0; i < 5; i++){
                prekazka = new Bod(OppRobots[i].Position.X, OppRobots[i].Position.Y);
                prekazka2 = new Bod(MyRobots[i].Position.X, MyRobots[i].Position.Y);
                if (IsObstacle(robo, prekazka, cil)){
                    lis.Add(new RobotDistance(new Bod(prekazka.x, prekazka.y), distanceTwoPoints(
                        robot, prekazka)));
                }
                if (IsObstacle(robo, prekazka2, cil) && i != j){
                    lis.Add(new RobotDistance(new Bod(prekazka2.x, prekazka2.y),
                        distanceTwoPoints(robot, prekazka2)));
                }
}
var sort = from s in lis orderby s.vzdalenost select s;
```

Výpis 9: Zjištění nejbližší překážky

Dle ukázky lze vidět, že při přidávání do listu se nerozlišuje, jestli se přidává robot ze stejného týmu, nebo z protivníkova. To proto, že nezáleží na tom kdo je tou překážkou, ale na tom, že překážka je a ta se musí objet. Po provedení tohoto kódu se veme první objekt z listu a dosadí se do výpočtu polynomu pro naplánování cesty. Pro tento postup je použit přibližně stejný kód, jako je ukázan ve výpise č. 7. Po kompletním zaimplementování pro všechny roboty nelze porovnávat novou verzi simulátoru se zaimplentovanou částí pro plánovaní cesty s původní verzi z hlediska vysledků, nebo efektivity, protože už při prvním objetí překážky se robot dostane do jiného gridu než v původní verzi a strategie pro tuto situaci vyhodnotí jiný vývoj hry. Ovšem porovnávat tyto dvě věrze lze z hlediska kolizí. Při testech byl měřen počet kolizí na několika různých strategiích

s touto implementací a následovně bez ní. V následující tabulce jsou ukázány výsledky testů.

	S_1xS_2	S_4xS_5	S_8xS_5	S_1xS_6	S_1xS_4	S_6xS_5	S_4xS_2	S_3xS_5	celkem
s regresí	2200	350	150	1000	1600	50	150	2800	8300
bez regrese	4500	1000	550	6400	3300	200	400	7900	24250

Tabulka 3: Počet kolizí v jednotlivých zápasech

Testy byly prováděny s osmi náhodně volenými strategiemi S_1-S_8 a každá testovací hra byla dlouhá 20 vteřin. V tabulce lze vidět, že počet kolizí s implementací regresní metody je vždy menší.

5 Závěr

Cílem této práce bylo naplánovat cestu robota tak, aby se dokázal vyhnout překážkám a dojel úspěšně ke svému cíli. Tento úkol se podařilo splnit, protože průměrný počet kolizí opravdu klesl a ve většině případech robot překážku v cestě dokáže objet. Oproti původní verzi simulátoru, klesl počet kolizí v průměru 3x. Při reálné hře toto znamená velké plus nejen z ekonomického hlediska, kdy se nám nebudou neustálým narážením do sebe ničit roboti, ale také při soutěžích, kdy vrážení do jiného robota je hodnoceno jako faul. V simulátoru hry se ale stále vyskytují situace, kdy ke kolizím dochází. Tento problém je z velké částí způsoben tím, že v simulátoru při hledání kolizí se považuje čtverec(robot) za kruh a tím dochází velmi často ke kolizím, i když k ním ve skutečnosti nedošlo. Například pokud jedou dva roboti blízko vedle sebe. Další situace, při které dochází stále ke kolizím nastává, pokud mají dva body (které se dosazují do výpočtu polynomu) stejnou x-ovou souřadnici. Polynom se poté vytváří jinak, než by se očekávalo. Tento problém by se mohl dále řešit záměnou os před výpočtem polynomu. Tím by se docílilo stejného výsledku, jako když se x-ové souřadnice liší. Bylo by vhodné pro další vývoj tohoto simulátoru se tomuto problému a kolizím dvou čtverců věnovat. Kolize a plánování cesty ale nejsou jediným problémem v simulátoru. Další vývoj by se měl také věnovat například chování útočníka, jenž nyní slepě následuje pouze pozici míče a neřeší soupeřovu bránu, nebo by se hodilo větší zapojení brankáře do hry a pomoct tak svým robotů v mnoha situacích.

Ondřej Gronych

6 Reference

[1] Lutus Paul, *Polynomial Regression*[online]. c2009, Poslední revize 3.10.2011 [cit. 15.12.2011]. Dostupné z http://www.arachnoid.com/sage/polynomial.html

[2] *LINQ Query Expressions*[online]. Poslední revize 12.2.2012 [cit. 18.4.2012]. Dostupné z http://msdn.microsoft.com/en-us/library/bb397676.aspx

[3] Šimeček, R. *Fotbal robotů*. Ostrava, 2007. 63s. Diplomová práce na Fakultě elektrotechniky a informatiky vysoké školy Báňské

[4] S. Keshmiri, S. Payandeh, Robot navigation controller: A non-parametric regression approach, Intelligent Control Systems, (2010)

[5] J. Fox, Nonparametric simple regression: smoothing scatterplots, Thousand Oaks, (2000)

[6] J. Fan, I. Gijbels, Local Polynomial Modelling and Its Applications, (1996)

[7] *Micro Robot World Cup Soccer Tournament(MiroSot)*[online]. Dostupné z http://www.fira.net/?mid=mirosot [cit. 23.4.2012]

7 Přílohy

I. Příloha na CD/DVD

Obsah CD/DVD

Adresář	Popis
Robot soccer	Aplikace simulátoru
Text	Text bakalářské práce